

Esta clase va a ser

- grabada

Clase 04. FUNDAMENTOS DE LA CIENCIA DE DATOS

Python Básico

Objetivos de la clase



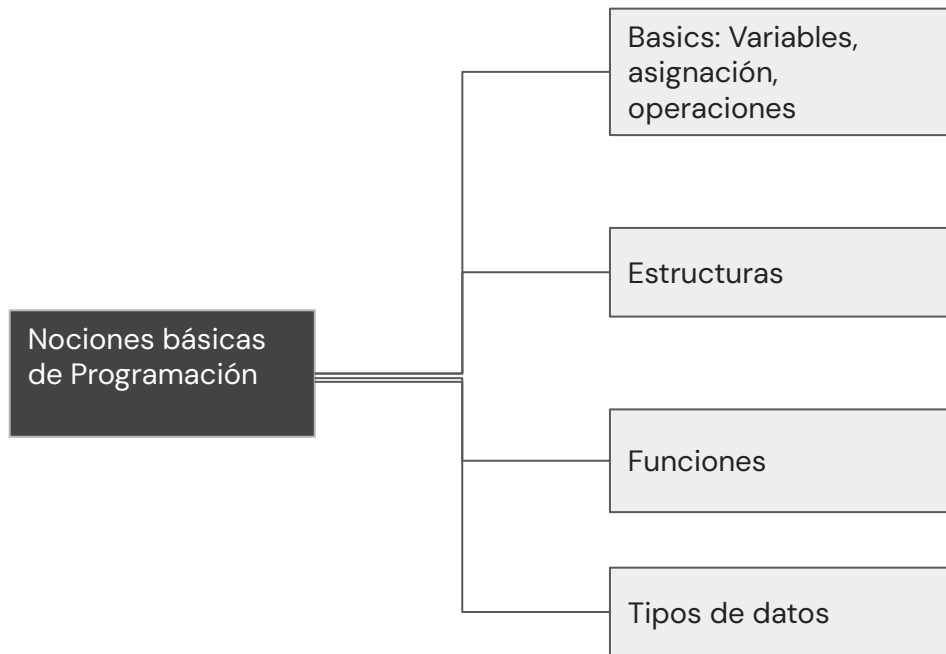
Conocer las distintas formas de desarrollo con Python



Comprender las nociones básicas de la programación estructurada.



MAPA DE CONCEPTOS



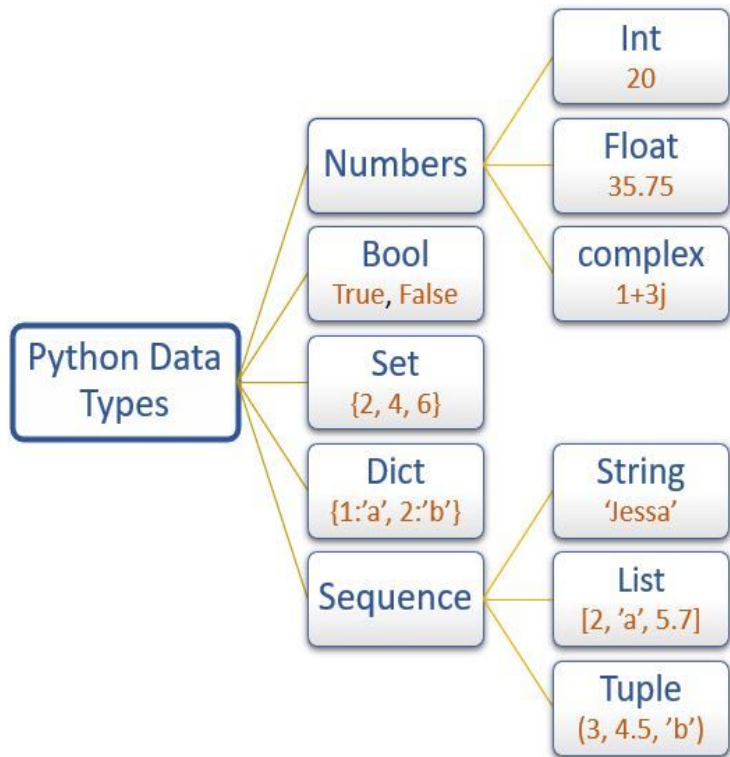
Introducción a la programación Python

- ✓ Definición de Programa (Lenguaje compilado vs interpretado)
- ✓ Introducción a Python
- ✓ Nociones básicas: variable asignación, expresiones
- ✓ Objetos y punteros
- ✓ Operadores: aritméticos, de asignaciones, de comparación y de identidad y pertenencia.

Tipos de datos en Python

Tipo de dato

- ✓ Define qué tipos de operaciones se puede hacer con él. Por ejemplo, un número se puede sumar, pero un texto no.
- ✓ Python define dos grandes grupos de tipos de datos: simples y estructurados.
- ✓ Podemos saber el tipo de un dato x con la función `type(x)`



Datos Simples

- ✓ Los tipos de datos **simples** están formados por un solo objeto de un solo tipo

Tipo	Ejemplo	Definición
int	x = 1	Enteros
float	x = 1.0	Punto flotante (decimales)
complex	x = 1 + 2j	Complejos (parte real e imaginaria)
bool	x = True	Booleanos o lógicos: verdadero / falso
str	x = 'abc'	Texto
NoneType	x = None	Tipo especial para indicar valores nulos

Datos Estructurados

- ✓ Los tipos de datos **estructurados** están formados por más de un objeto.
- ✓ El más utilizado es **list**, pero no es la única forma de trabajar con este tipo de datos.

Tipo	Ejemplo	Definición
list	[1, 2, 3]	Lista ordenada
tuple	(1, 2, 3)	Lista ordenada inmutable
dict	{'a':1, 'b':2, 'c':3}	Diccionario: conjunto de pares clave:valor
set	{1, 2, 3}	Conjunto, a la manera de un conjunto matemático

Mutabilidad

Mutabilidad

- ✓ La estructura list es **mutable** porque permite que sus elementos sufran modificaciones una vez definida.
- ✓ Por otro lado, las estructuras **inmutables** como las tuplas (tuple) no admiten esta reasignación de elementos en tiempo de ejecución del programa.

- ✓ La estructura dict, por su parte, es **mutable en sus valores**. Sin embargo, es **inmutable en sus claves**.

Estructuras de control

Estructuras de control: FOR, WHILE, IF

¿Qué son y para qué sirven?

- ✓ Las estructuras de control sirven para **dar claridad y orden al código.**
- ✓ Si hay que hacer operaciones repetitivas, estas estructuras nos ayudan a organizarlas.

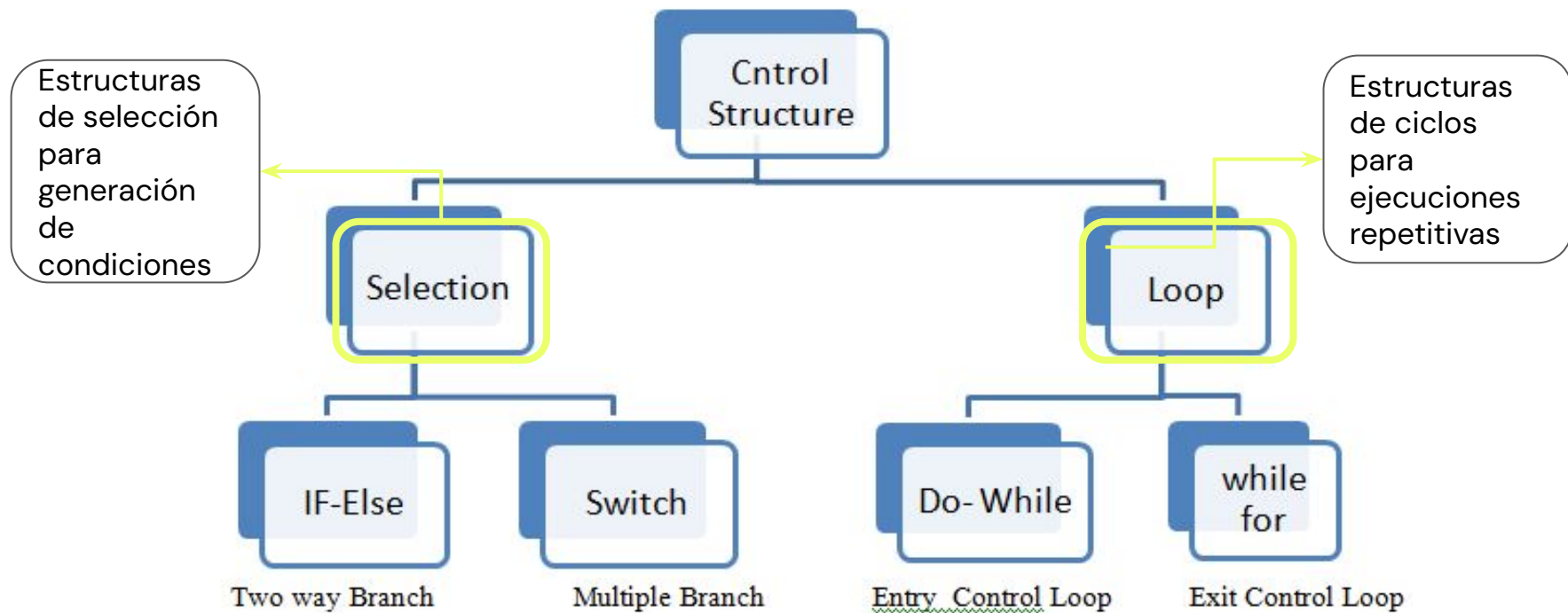
- ✓ Las estructuras de control más comunes son:

👉 For

👉 While

👉 If

👉 Switch (Otros lenguajes e.g C)

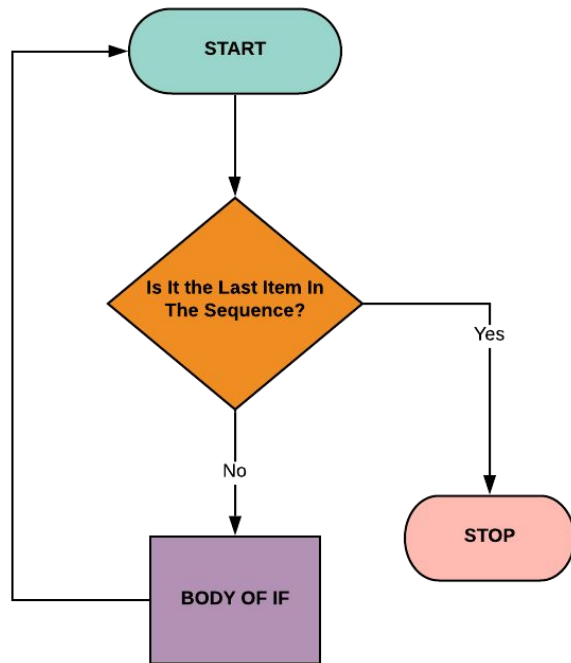


Estructura condicional FOR

Estructura FOR

✓ Repite un comando **una cantidad fija** de veces

```
for i in range(1,10):  
    print(i)                # muestra los números del 1 al 9  
  
for i in [1,4,6,2]:  
    print(i)                # muestra los números de la lista
```

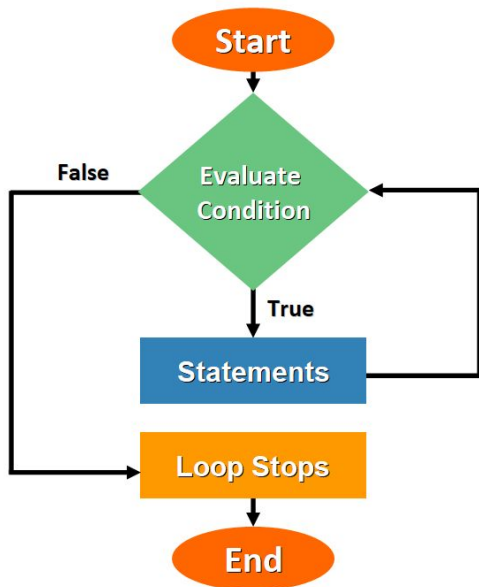


Estructura condicional

WHILE

Estructura WHILE

- ✓ Repite una secuencia de comandos **"mientras"** una **condición se cumpla**. Cuando la condición no se cumple más, termina la repetición.



```
i = 1
while i < 10: # el código luego de los dos puntos se ejecuta
    print(i)  # mientras i es menor a 10.
    i += 1    # cuando i llega a 10 termina la ejecución
```

Estructura condicional

IF

Estructura condicional (IF)

- ✓ **Si se cumple** una condición, se ejecuta una secuencia de comandos. **En otro caso**, se ejecuta otra.
- ✓ Pueden manejarse **más de dos opciones**.

```
x = 1
if x < 10:                                # Pregunto si x es menor a 10
    print(x, "es menor a 10") # Si es así muestro mensaje
elif x > 10:                              # Si no es así, pregunto si x es
    print(x, "es mayor a 10") # a 10 y si es así muestro mensaje
else:                                     # Si nada de lo anterior se
    print(x, "es 10")                 # cumple, ejecuto esto
```

Estructura condicional (IF)

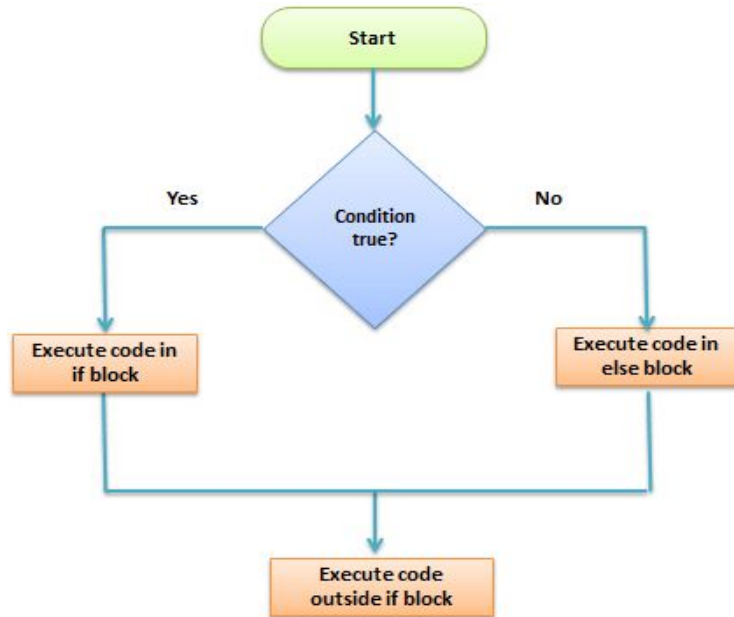
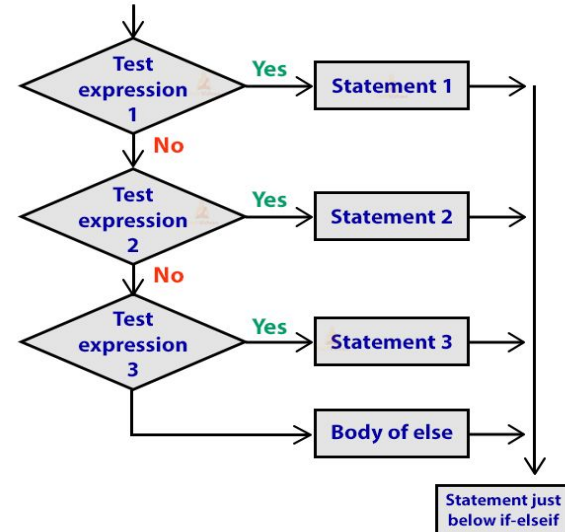


Diagrama de flujo condicionales

Python if-elif ladder



Estructura if-elif-else Python



Actividad colaborativa

Probando estructuras en Python

Deberán resolver en grupo dos problemas reales, utilizando las estructuras aprendidas de programación en Python en una notebook.

Duración: **20 minutos**

Optimizando el stock para una PYME

Consigna: Se tiene una lista con Valores=[200, 225, 232, 221, 243, 256, 255] que representan los precios de una acción de la compañía X la semana pasada (cada dato representa el promedio diario)
Escribir el código para calcular los días de la semana donde hubo un retroceso respecto al día anterior en el valor de la acción de la compañía X.



Sugerencia: Crear otra lista

```
Dias=['Lunes','Martes','Miercoles','Jueves','Viernes','Sabado','Domingo']
```

para hacer la iteración y utilice un ciclo con la siguiente estructura

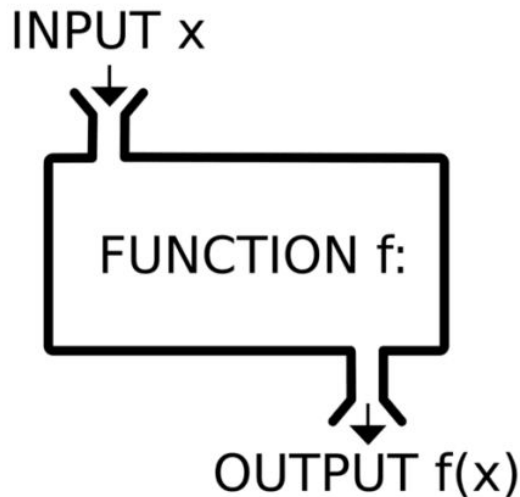
```
for x,y in zip(Dias,Valores):
```

 hacer uso de la función `np.diff` y de condicionales

Funciones, argumentos y retorno

Funciones

Funciones



- ✓ Para trabajar profesionalmente en programación, el código que se usa en forma repetitiva se organiza en funciones.
- ✓ Puede hacerse una analogía con una función matemática $y = f(x)$: la función f recibe un argumento x , ejecuta una serie de comandos y devuelve un valor y .

Argumentos y retorno

Argumentos y retorno

Las funciones tienen al menos 3 elementos:

- 👉 El nombre de la función
- 👉 Cero o más argumentos (variables de entrada)
- 👉 Un valor de retorno (salida de la función)

```
def add(x, y):  
    print(f'arguments are {x} and {y}')    return x + y
```

1. def keyword

2. function name

3. function arguments inside ()

4. colon ends the function definition

5. function code

6. function return statement

Sintaxis de una función

```
print(x)          # función nativa de Python que muestra el valor de x
print(x,y)        # print puede mostrar los valores de más de una variable

def suma(x,y):    # aquí definimos una función propia con argumentos x e y
    z = x + y     # la función suma los valores x e y, y asigna resultado a z
    return z      # el valor de retorno es z

res = suma(2,3)   # aplicamos la función definida a los números 2 y 3
                  # y guardamos el resultado en res
print (res)       # mostramos el resultado: 2 + 3 = 5
```

Ejemplo

```
def suma(x,y):    # Aquí definimos una función "suma".
    z = x + y     # Esto es lo que pide el ejercicio
    return z

res = suma(2,3)   # Aquí probamos la función suma con dos números concretos
                  # Esta es la prueba para verificar que el código funciona
print (res)
```



Ejemplo en vivo

¿Cómo el uso de funciones permite resolver un problema real?

10 minutos

Ejemplo aplicado de funciones

- ✓ ¿Cuál sería el **valor esperado al final** de la semana para el inversor?
- ✓ Crear una función llamada **retorno_semanal** que calcule el valor esperado con la **cantidad de acciones compradas cada día, probabilidad de ganancia y no ganancia**

El precio de la acción cada día entre

Lunes-Domingo es: `Valores= [200, 225, 232, 221, 243, 256, 255]`

Pueden utilizar la siguiente fórmula:

$$\mathbb{E}[X] = \sum_{i=1}^n x_i P(X = x_i)$$

Ejemplo aplicado de funciones

Un inversor financiero está interesado en invertir en la compañía. Se propone comprar durante **cada día de la semana 20 acciones**.

También sabe que la **probabilidad de obtener ganancias (aproximadamente 15% en cada inversión)** es de **0.56** y la **probabilidad de perder el 18%** es **0.44** (Solo hay esas dos opciones).



Break

¡10 minutos y volvemos!

IPython, trabajo con Notebooks

Creando un Jupyter Notebooks

IPython y notebooks

La clase pasada hablamos un poco sobre los notebooks. Veamos un poco más a detalle el tema 🚀

Para iniciar el cuaderno jupyter, se debe escribir el siguiente comando en la terminal:

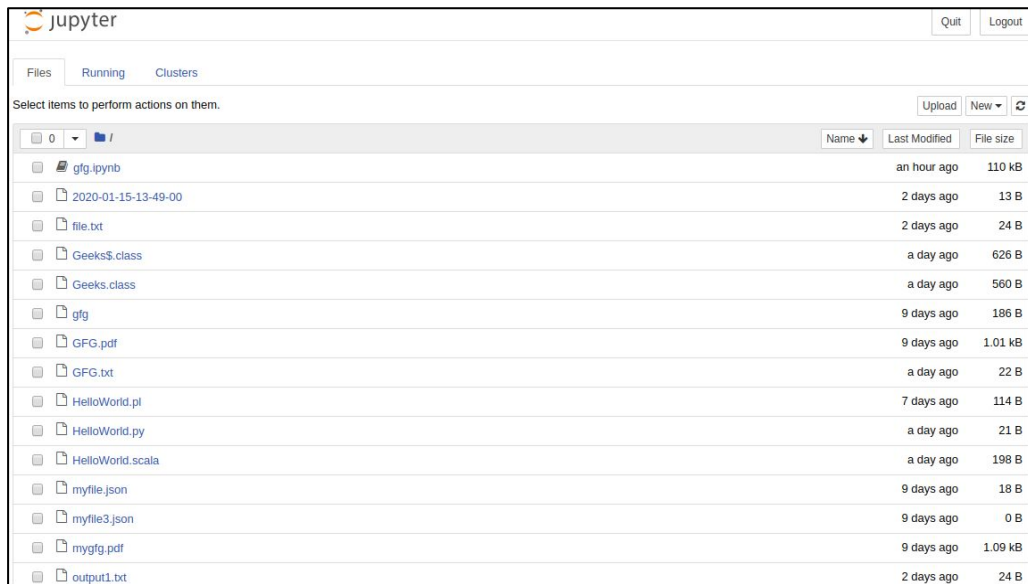
```
jupyter notebook
```

Esto imprimirá cierta información sobre el notebook server en su terminal, incluida la URL de la aplicación web (de forma predeterminada, `http://localhost:8888`) y luego abrirá su navegador web predeterminado a esta URL.

IPython y notebooks

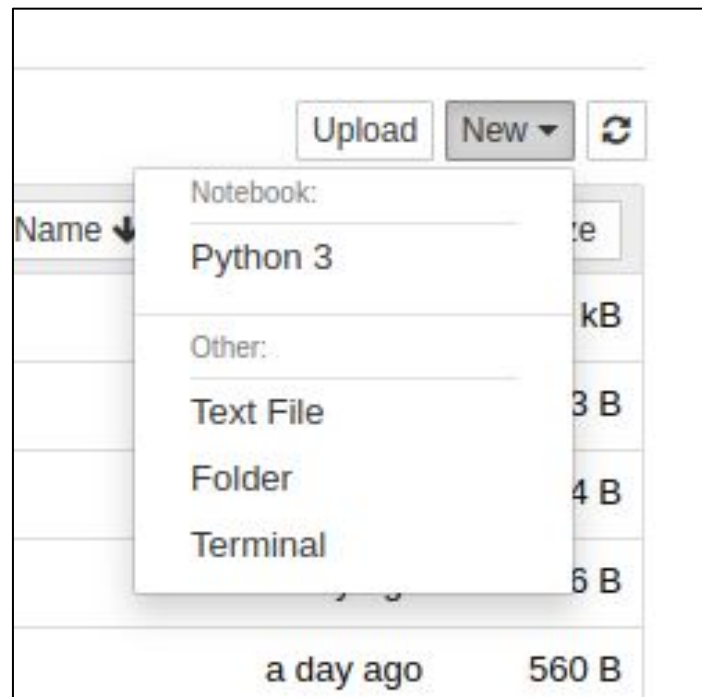
Una vez que se abre, verán un panel, que mostrará una lista de archivos y subdirectorios en el directorio donde se inició el servidor de la libreta.

La mayoría de las veces, desearía iniciar un servidor de notebooks en el directorio de nivel más alto que contenga cuadernos. A menudo, este será su directorio de inicio.

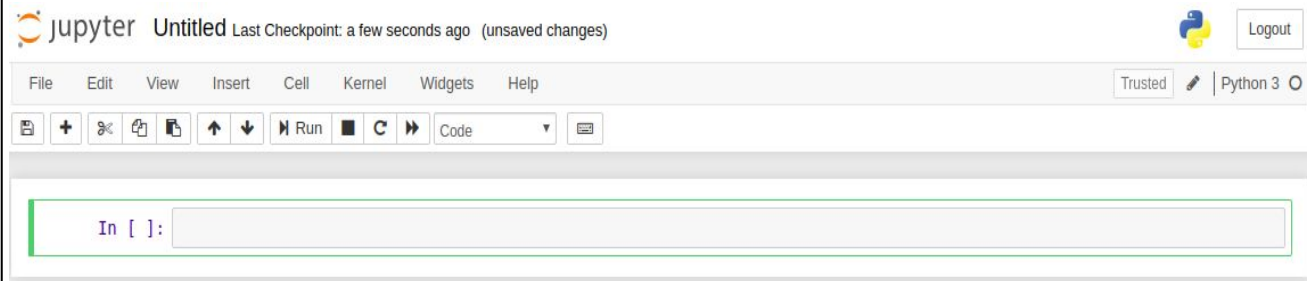
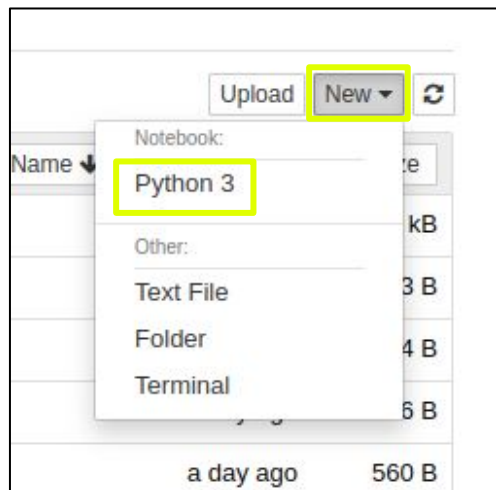


IPython y notebooks

Para crear un nuevo cuaderno, haga clic en el botón nuevo (New) en la esquina superior derecha. Haga clic en él para abrir una lista desplegable y luego, si hace clic en Python3, se abrirá un nuevo cuaderno.



IPython y notebooks



Celdas

Celdas

Las celdas pueden considerarse como el cuerpo del Jupyter.

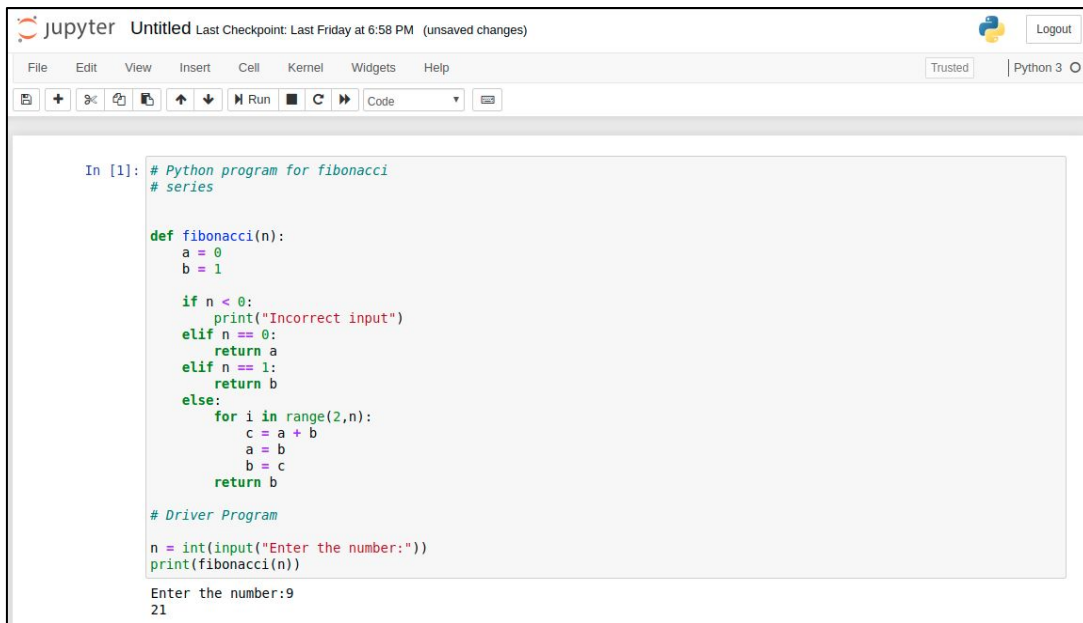
Existen tres tipos de celdas:

1. Código
2. Markdown
3. Raw NBConverter



1. Código

Es donde se escribe el código y, cuando se ejecuta, el código mostrará la salida debajo de la celda. En este ejemplo se crea un código simple de la serie Fibonacci.



The screenshot shows a Jupyter Notebook window titled "Untitled" with a last checkpoint from "Last Friday at 6:58 PM" and "(unsaved changes)". The interface includes a top menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The code cell contains the following Python code:

```
In [1]: # Python program for fibonacci
# series

def fibonacci(n):
    a = 0
    b = 1

    if n < 0:
        print("Incorrect input")
    elif n == 0:
        return a
    elif n == 1:
        return b
    else:
        for i in range(2,n):
            c = a + b
            a = b
            b = c
        return b

# Driver Program

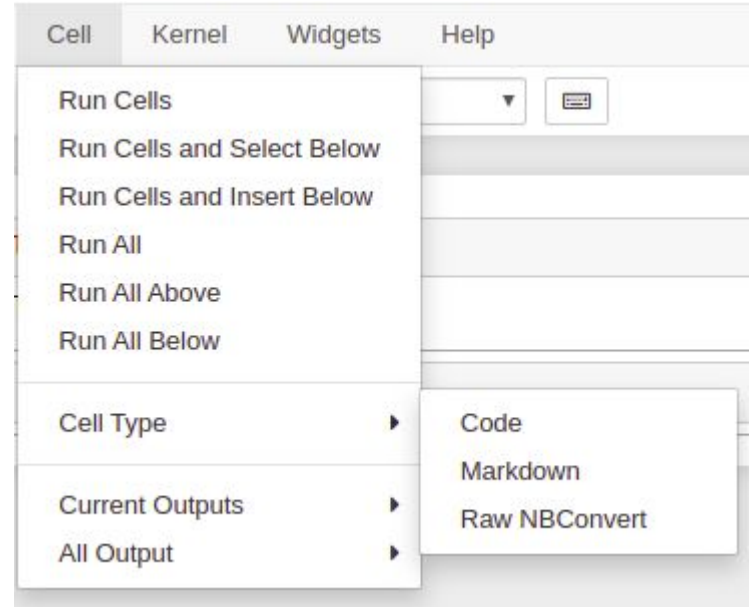
n = int(input("Enter the number:"))
print(fibonacci(n))
```

Below the code, the input and output are shown:

```
Enter the number:9
21
```

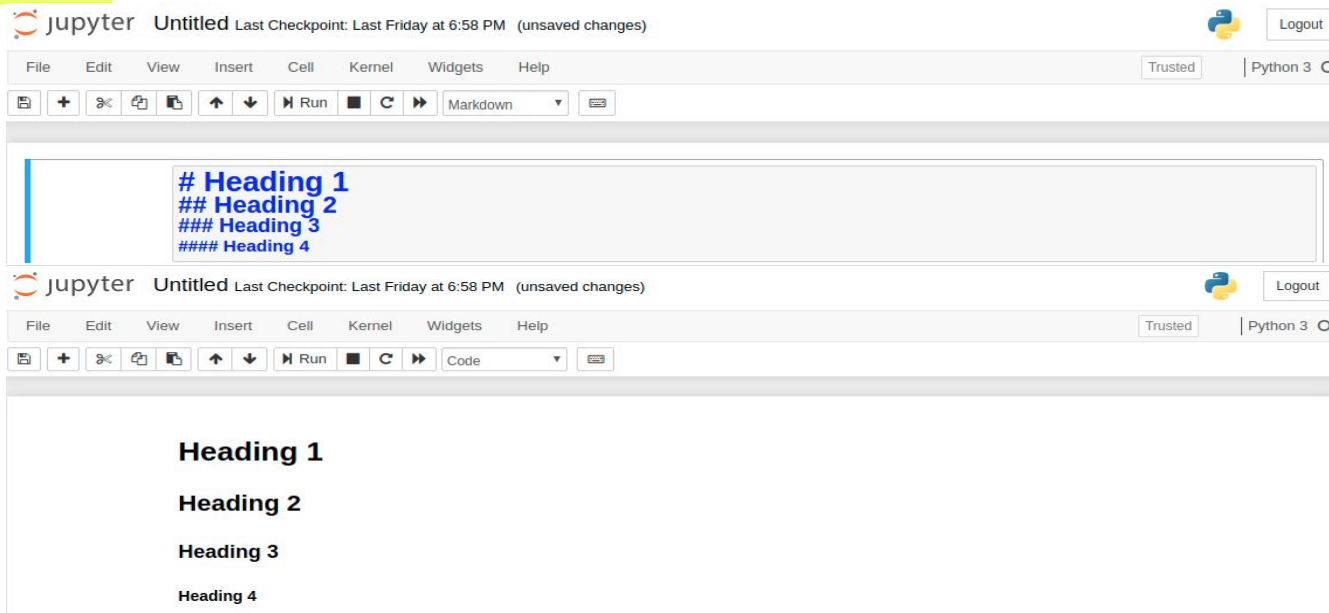
2. Markdown

Markdown es un lenguaje de marcado popular que es el superconjunto del HTML. Jupyter Notebook también admite rebajas. El tipo de celda se puede cambiar.



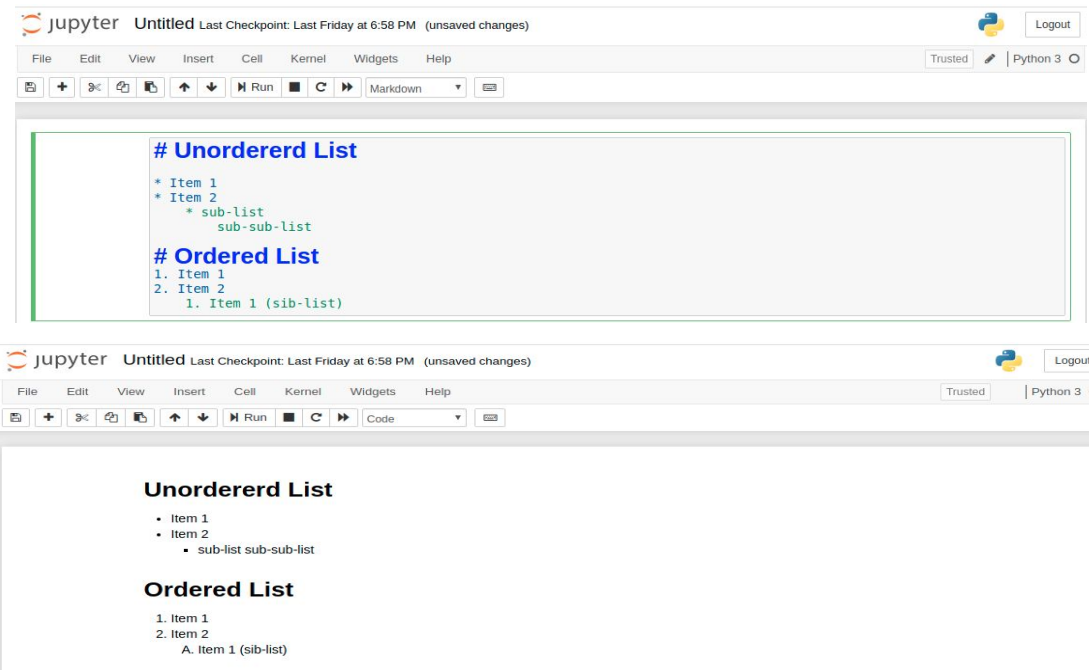
2. Markdown

El encabezado se puede agregar anteponiendo cualquier línea con un '#' único o múltiple seguido de un espacio.



2. Markdown

Se puede agregar Orden de Jerarquía usando el signo '*'.



The image displays two screenshots of the Jupyter Notebook interface, illustrating the rendering of Markdown code into a visual hierarchy.

Top Screenshot (Code View): The notebook is titled "Untitled" and shows a code cell with the following Markdown content:

```
# Unordererd List

* Item 1
* Item 2
  * sub-list
    sub-sub-list

# Ordered List

1. Item 1
2. Item 2
  1. Item 1 (sib-list)
```

Bottom Screenshot (Rendered View): The same notebook is shown after rendering the code. The output displays the visual hierarchy:

Unordererd List

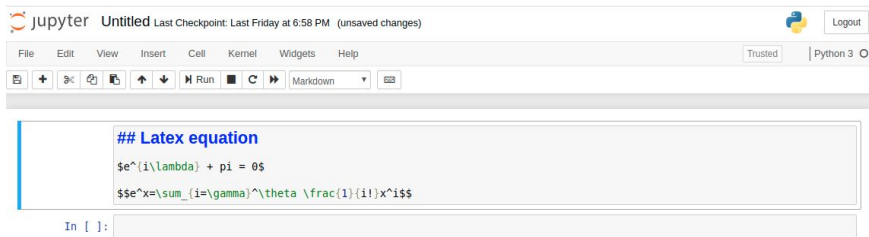
- Item 1
- Item 2
 - sub-list sub-sub-list

Ordered List

1. Item 1
2. Item 2
 - A. Item 1 (sib-list)

2. Markdown

Permite añadir Ecuaciones en formato Latex y tablas



The screenshot shows a Jupyter Notebook interface with the title "Untitled" and a subtitle "Last Checkpoint: Last Friday at 6:58 PM (unsaved changes)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The toolbar shows icons for file operations, running, and cell types. The code cell contains the following LaTeX code:

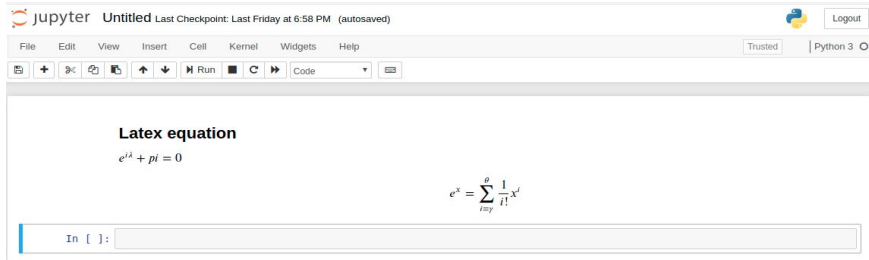
```
## Latex equation

$$e^{i\lambda} + p_1 = 0$$


$$e^x = \sum_{i=0}^{\infty} \frac{\gamma^i}{i!} \theta \frac{1}{i!} x^i$$

```

Below the code cell is an input prompt "In []:".

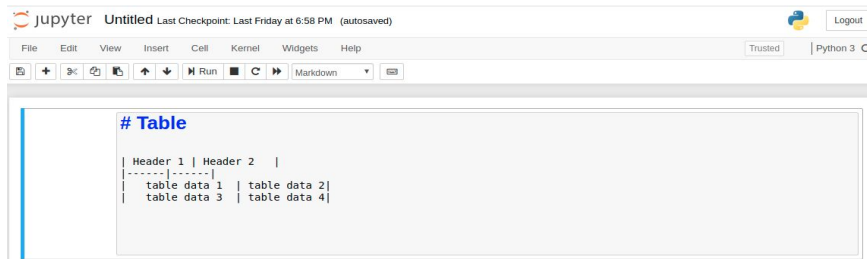


The screenshot shows the same Jupyter Notebook interface, but the code cell has been executed. The output is displayed below the code cell:

Latex equation

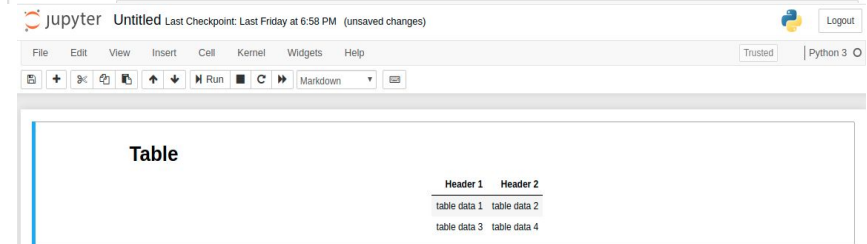
$$e^{i\lambda} + p_1 = 0$$
$$e^x = \sum_{i=0}^{\infty} \frac{1}{i!} x^i$$

Below the output is an input prompt "In []:".



The screenshot shows a Jupyter Notebook interface with the title "Untitled" and a subtitle "Last Checkpoint: Last Friday at 6:58 PM (autosaved)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The toolbar shows icons for file operations, running, and cell types. The code cell contains the following Markdown code for a table:

```
# Table
| Header 1 | Header 2 |
|-----|-----|
| table data 1 | table data 2 |
| table data 3 | table data 4 |
```



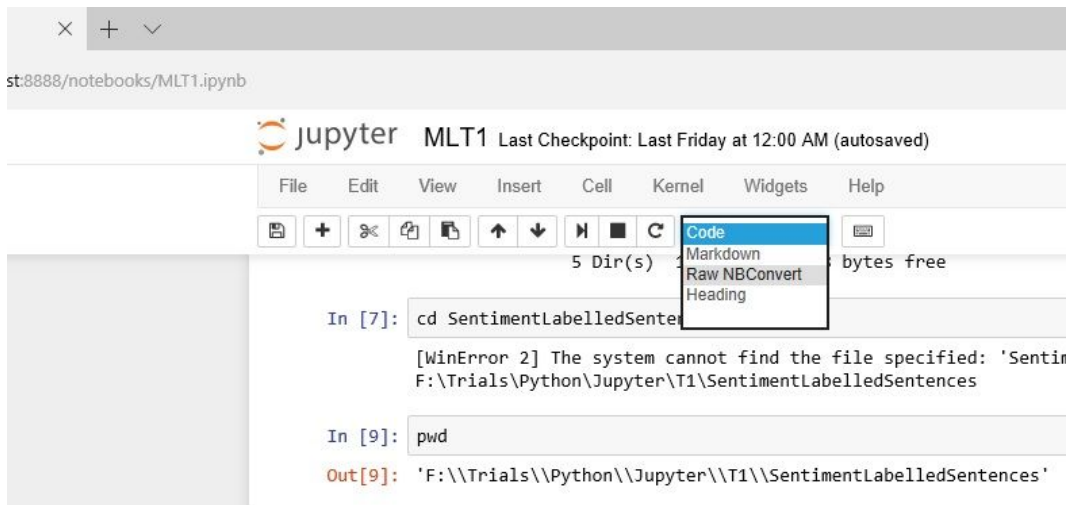
The screenshot shows the same Jupyter Notebook interface, but the code cell has been executed. The output is displayed below the code cell:

Table

Header 1	Header 2
table data 1	table data 2
table data 3	table data 4

3. Raw NBConverter

Se proporcionan celdas sin procesar para escribir la salida directamente. Esta celda no es evaluada por el cuaderno de Jupyter. Después de pasar por nbconvert, las celdas sin formato llegan a la carpeta de destino sin ninguna modificación.

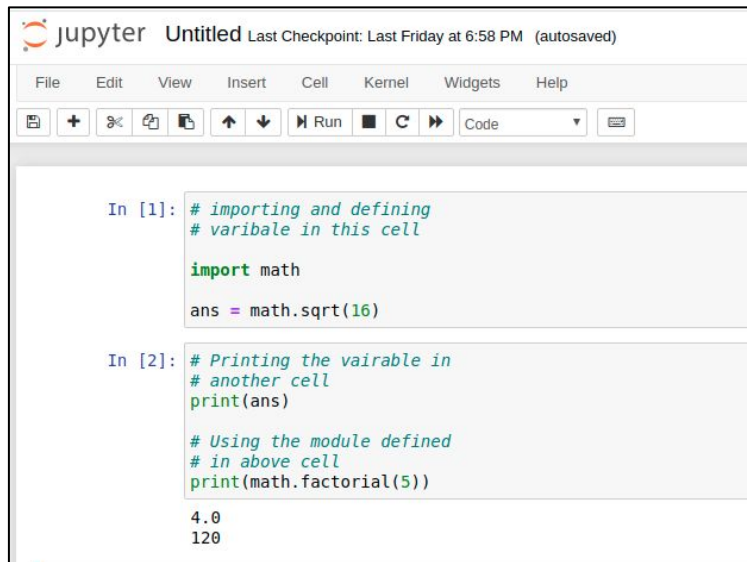


Kernel

Kernel

Un kernel se ejecuta detrás de cada notebook. Siempre que se ejecuta una celda, el código dentro de la celda se ejecuta dentro del kernel y la salida se devuelve a la celda para que se muestre.

Entonces, si se importa un módulo en una celda, ese módulo estará disponible para todo el documento, por ejemplo:



```
Jupyter Untitled Last Checkpoint: Last Friday at 6:58 PM (autosaved)
File Edit View Insert Cell Kernel Widgets Help
+ %< [Run] [Code]

In [1]: # importing and defining
        # varibale in this cell

        import math

        ans = math.sqrt(16)

In [2]: # Printing the vairable in
        # another cell
        print(ans)

        # Using the module defined
        # in above cell
        print(math.factorial(5))

4.0
120
```

Kernel

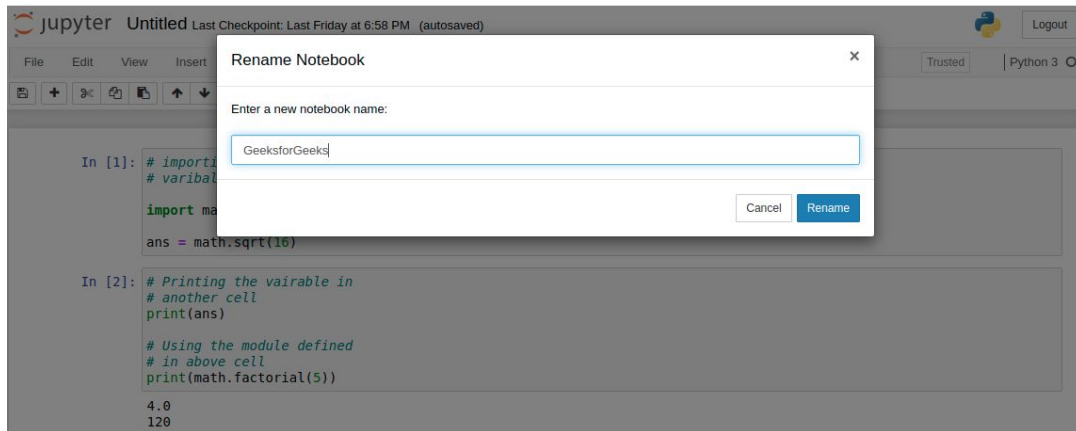
Jupyter Notebook ofrece varias opciones para kernels. Esto puede ser útil si se desea restablecer cosas. Las opciones son:

1. Reiniciar: Esto reiniciará los núcleos, es decir, borrará todas las variables que se definieron, borrará los módulos que se importaron, etc.
2. Reiniciar y borrar el output: Hará lo mismo que "Reiniciar", pero también borrará toda la salida que se mostró debajo de la celda.
3. Reiniciar y ejecutar todo: Esto hará lo mismo que "Reiniciar y borrar el output", pero también ejecutará todas las celdas en el orden de arriba hacia abajo.
4. Interrumpir: Esta opción interrumpirá la ejecución del kernel. Puede ser útil en el caso de que los programas continúen para su ejecución o si el kernel se atasca en algún cálculo.

Untitled.ipynb

Cuando se crea el cuaderno, Jupyter Notebook nombra el cuaderno como Untitled.ipynb de forma predeterminada.

Para cambiar el nombre del cuaderno, simplemente haga clic en la palabra Untitled.ipynb. Esto abrirá un cuadro de diálogo titulado "Cambiar nombre del cuaderno". Ingrese el nombre válido para su cuaderno en la barra de texto, luego haga clic en Aceptar.



Instalación de Jupyter notebooks, uso de **Google Colab**

Jupyter Notebooks

Jupyter Notebook es una aplicación cliente-servidor lanzada en 2015 por la organización sin ánimo de lucro [Proyecto Jupyter](#). Permite crear y compartir documentos web en formato JSON.

Tiene celdas que permiten almacenar código, texto (en formato Markdown), fórmulas matemáticas y ecuaciones, o también contenido multimedia (Rich Media).

Se ejecuta desde la aplicación web cliente que funciona en cualquier navegador estándar.

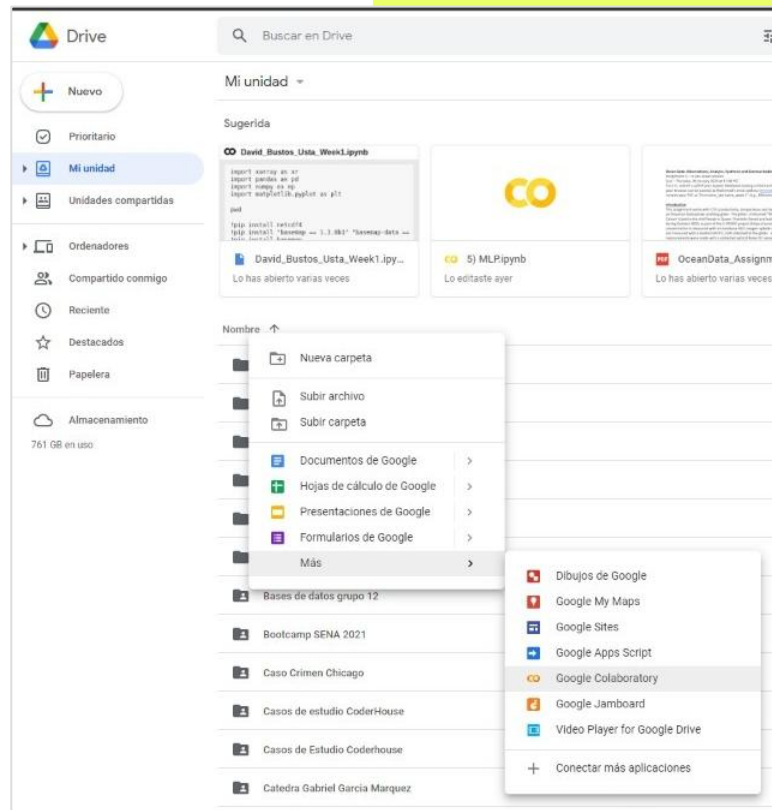
El requisito previo es instalar y ejecutar en el sistema el servidor Jupyter Notebook por medio de Anaconda (ver [Instalacion Anaconda](#)).

Los documentos creados en Jupyter pueden exportarse, entre otros formatos, a HTML, PDF, Markdown o Python (-py o .ipynb) y también pueden compartirse con otros usuarios por correo electrónico, utilizando Dropbox o GitHub

Google Colab

Para utilizarlo basta con acceder a nuestra cuenta de Google y, o bien entrar directamente al enlace de [Google Colab](https://colab.research.google.com/).

Otra opción es acceder a Google Drive, pulsar el botón de «Nuevo» y desplegar el menú de «Más», luego seleccionar «Colaboratory» y crear un nuevo cuaderno (notebook).



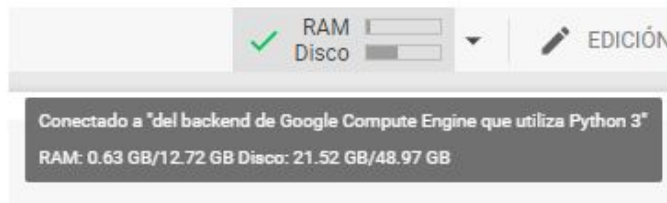
Google Colab

Cuando creamos un nuevo cuaderno, este es «estático», es decir, vemos su contenido, pero no estamos conectados a ningún entorno de ejecución.

Nuestro cuaderno se conecta a una VM de Google Compute Engine (la infraestructura de máquinas virtuales de Google en la nube) cuando ejecutamos una celda o pulsamos sobre el botón de «Conectar».



La máquina en un inicio cuenta con 12 GB de RAM y 50 GB de almacenamiento en disco disponibles para el uso.





Actividad colaborativa

Cálculo de estadística descriptiva básica

Aplicando conceptos de programación estructurada para obtener resúmenes numéricos

Duración: **20 minutos**

Optimizando el stock para una PYME

Consigna: Utilizaremos la información de la Clase 3 asociada con las acciones de diversas compañías para resolver las siguientes consignas:

👉 Por medio de un ciclo (For o While) obtener el promedio, desviación estándar y varianza de cada una de las acciones en cada columna

👉 Crear una función que itere sobre cada columna de las acciones e identifique valor maximo y minimo
Hint: Pueden usar las funciones `.mean()`, `.std()`, `.var()`, `.min()`, `.max()` de Pandas

Se recomienda trabajar en grupos de 2 o 3 estudiantes.



Elección de Datasets potenciales

Deberás entregar el primer avance de tu proyecto final. Identificarás 3 datasets potenciales con las siguientes características: i) al menos 2000 filas, ii) al menos 15 columnas. Posterior a esto crearás un notebook donde cargarás los datos utilizando la librería Pandas y finalmente describirás las variables que sean más interesantes teniendo en cuenta el contexto comercial y analítico del problema que se quiera resolver.



Datasets con la librería Pandas

Consigna

- ✓ Identificar 3 datasets que cumplan con las siguientes condiciones: a) al menos 2000 filas y b) al menos 15 columnas. Pueden buscar en las siguientes fuentes: Github, Kaggle, Google Dataset Search (también puede ser un archivo propio).
- ✓ Crear una cuenta de GitHub y hostearlo ahí.
- ✓ Cargar los archivos correspondientes por medio de la librería Pandas.
- ✓ Describir las variables potencialmente interesantes en cada archivo teniendo en cuenta el contexto del caso..

Aspectos a incluir

- ✓ El dataset debe estar subido a Github y compartido desde allí.
- ✓ El código debe estar hecho en un notebook y debe estar probado.



Datasets con la librería Pandas

Formato

- ✓ Entregar un archivo con formato .ipynb.
Debe tener el nombre
"Datasets+Apellido.ipynb".

Sugerencias

- ✓ Preparar el código y probar los
resultados con distintas entradas

¿Preguntas?



¿Quieres saber más?
**Te dejamos material
ampliado de la clase**



MATERIAL AMPLIADO

Recursos multimedia



[Guia de instalación Anaconda](#)

Disponible en nuestro repositorio.

CLASE N°4

Glosario

Estructura de control: nos permiten hacer operaciones repetitivas y nos dan orden y claridad en el código, se dividen en selección y cíclicas, mientras que las más comunes son (for, while, if)

Estructuras de selección: son aquellas que nos permiten generar condiciones para una operación

Estructuras cíclicas: nos permiten realizar operaciones repetitivas para una operación

For: estructura cíclica que repite varias veces una tarea (hacer hasta)

While: estructura de control cíclica que permite hacer hasta que se cumpla una condición de pare (hacer mientras)

If: estructura de selección que permite extraer condiciones de interés

Funciones: estructuras de programación con nombre propio que reciben argumentos y arrojan un resultado

Tipos de datos: estructuras básicas que permiten realizar operaciones (e.g. números, bool, diccionarios, tuplas, listas, strings)

Mutabilidad: cualquier objeto que sea susceptible de modificar sus valores luego de ser creado

Tipos de celdad Jupyter notebook: pueden ser de tres tipos: 1) código 2) Markdown (para texto y opciones HTML) y 3) RawNBConverter (sin procesar que no tienen efecto en el código)

Muchas gracias.

Resumen de la clase hoy

- ✓ Nociones básicas: Estructuras de Control, Operadores y Funciones.
- ✓ Tipos de datos
- ✓ Ipython trabajo con notebooks
- ✓ Instalación de Jupyter notebooks, uso de Google Colab

#DemocratizandoLaEducación